VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018 Lukáš Dočkal

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Dočkal**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: IDC CEMA, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Konzultant bakalářské práce: Jan Mikurda

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2018 Dožbal .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2018 .....

Na tomto místě bych chtěl poděkovat všem z IDC CEMA za umožnění absolvování bakalářské praxe a také bych pak chtěl poděkovat kolegům z týmu Query Tool za pomoc a odborné vedení při zpracovávání této práce. Dále bych pak chtěl poděkovat panu Ing. Davidu Ježkovi, Ph.D. za jeho rady při tvorbě této práce.

Abstrakt

Tato bakalářská práce popisuje vykonávanou praxi ve firmě IDC CEMA. Zmíněná firma je jedna z nejvýznamnějších světových analytických a poradenských společností. Poskytuje průzkumy a analýzy trhu, předpovídá vývoj a trendy v oblasti informačních a komunikačních technologií (ICT) a pořádá odborné konference. V rámci této bakalářské praxe jsou řešeny problémy týkající se webové aplikace Query Tool, která je využívána analytiky a klienty společnosti IDC.

Klíčová slova: Query Tool, IDC, ICT, webová aplikace

Abstract

This bachelor thesis describes the practice in the company IDC CEMA. The company is one of the world's leading analytical and consulting companies. Provides market research and analysis, predicts developments and trends in Information and Communication Technologies (ICT), and organizes expert conferences. This bachelor's practice addresses issues related to Query Tool web applications, which are used by analysts and IDC clients.

Key Words: Query Tool, IDC, ICT, web aplication

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

1 Úvod **10**
1.1 Odborné zaměření společnosti 10
1.2 Popis pracovní pozice 10

2 Seznam zadaných úkolů a vyjádření jejich časové náročnosti **11**

3 Použité technologie **12**
3.1 Java 12
3.2 Spring 12
3.3 Javascript 12
3.4 PostgreSQL 13
3.5 Hibernate 13
3.6 HTML 13
3.7 CSS 13
3.8 RabbitMQ 14

4 Práce na projektu Query Tool **15**
4.1 Popis projektu 15
4.2 Blokování podezřelých uživatelů 16
4.3 Omezení přístupu do apliakce Query Tool 18
4.4 Zakázaní publikace pro dané prostředí 20
4.5 Tvorba katalogu produktů 22
4.6 Tvorba uživatelského rozhraní pro nastavení preferovaných měn 23
4.7 Zefektivnění transformací uživatelských dat 25

5 Znalosti a dovednosti uplatněné v průběhu odborné praxe **29**

6 Závěr **30**
6.1 Scházející znalosti a dovednosti v průběhu odborné praxe 30
6.2 Celkové zhodnocení odborné praxe a dosažených výsledků 30

Literatura **31**

Seznam použitých zkratek a symbolů

IDC	–	International Data Corporation
IDG	–	International Data Group
SQL	–	Structured Query Language
JSP	–	JavaServer Pages
AWS	–	Amazon Web Services
Scrum	–	Iterativní a inkrementální metodologie agilního vývoje
HTML	–	Hyper Text Markup Language
API	–	Application Programming Interface
REST	–	Representational State Transfer
UI	–	User Interface
XML	–	Extensible Markup Language
DEV	–	Development Environment
PROD	–	Production Environment
QA	–	Quality Assurance
MIRROR	–	Zradlené prostředí vzhledem k prosředí DEV
JSON	–	JavaScript Object Notation
WTFPL	–	Do What The F*ck You Want To Public License
IT	–	Informační Technologie

Seznam obrázků

1 Graf zobrazující počet zadaných úkolů 11
2 Graf zobrazující přibližný odpracovaný čas v hodinách 11
3 Ukázka UI aplikace Query Tool 15
4 Ukázka UI rozhraní pro blokování uživatelů 17
5 Model uživatelského rozhraní pro nastavení preferovaných měn 24
6 Finální verze pro nastavení preferovaných měn 24
7 Ukázka formuláře pro přidání dat 26

1 Úvod

Cílem této práce je popsat vykonávání odborné praxe ve firmě IDC CEMA.

1.1 Odborné zaměření společnosti

IDC [9] je předním světovým poskytovatelem zpravodajství trhu, poradenských služeb a událostí pro trhy informačních technologií, telekomunikací a spotřebitelských technologií. S více než 1 100 analytiky po celém světě společnost IDC nabízí globální, regionální a místní odborné znalosti o technologiích, příležitostech a trendech v průmyslu ve více než 110 zemích. Analýza a vhled společnosti IDC pomáhá IT profesionálům, manažerům v podnikání a také investičním společnostem činit rozhodnutí o technologických faktech a dosáhnout jejich klíčových obchodních cílů.

Společnost IDC, která byla založena v roce 1964, je dceřinou společností IDG, přední světové společnosti zabývající se sdělovacími, datovými a marketingovými službami.

1.2 Popis pracovní pozice

Ve firmě jsem nastoupil na pozici Java vývojáře do týmu Query Tool, který vyvíjí stejnojmennou aplikaci Query Tool používanou především zákazníky a analytiky. Mojí prací bylo podílení se na vývoji a údržbě nových funkcionalit pro aplikaci Query Tool. Tým, ve kterém jsem pracoval, se skládal z pěti členů a využíval agilní metodiku Scrum.

Celý sprint trval dva týdny. Každý den jsme měli společně stand up, který vedl scrum master (vedoucí týmu) a postupně každý z nás krátce shrnul práci v předešlém dnu a co se chystáme dále dělat. Po každém sprintu následovala reflexe (ohlédnutí za předešlým sprintem), kde si každý z nás připravil pár vět na téma: co bylo splněno, co je potřeba zlepšit a co jsme naučili. Úkoly, které jsem zpracoval během praxe, jsou podrobně popsány v kapitole seznam úkolů a jejich řešení.

2 Seznam zadaných úkolů a vyjádření jejich časové náročnosti

1. - 6. den Seznámení se s prostředím a instalace potřebných programů

- První den ve společnosti IDC jsem byl seznámen s používanými technologiemi. Následně v dalších dnech jsem instaloval programy potřebné pro práci.

6. - 17. den Seznámení se s projektem Query Tool

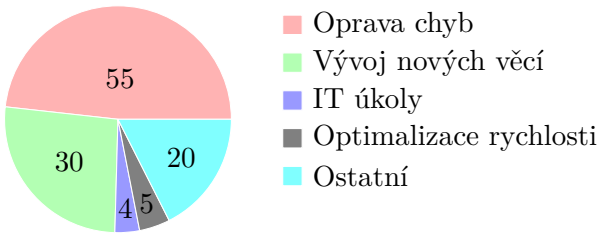
- V tomto období jsem absolvoval několik školení, které měly za úkol mi blíže představit fungování společnosti a také představení projektu Query Tool, na kterém jsem pracoval podobu celé odborné praxe.

18. - 33. den Práce na jednodušších úlohách

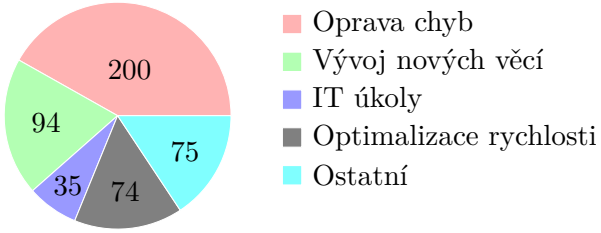
- V těchto dnech jsem byl zapojen do vývoje aplikace Query Tool prostřednictvím jednoduchých úloh. Tyto se týkaly práce především na klientské straně aplikace. Jednalo se konkrétně o úpravu JPS souborů a JS skriptů. V tomto období jsem se seznámil s agilní metodikou Scrum, kterou tým využíval.

33. - 50. den Práce na komplexních úlohách

- Poslední období odborné praxe jsem řešil komplexní úlohy, které se týkaly především vývoje nových funkcionalit. Naučil jsem se také spoustu postupů při implementaci požadavků a také týmového vývoje.



Obrázek 1: Graf zobrazující počet zadaných úkolů



Obrázek 2: Graf zobrazující přibližný odpracovaný čas v hodinách

3 Použité technologie

3.1 Java

Java [6] je objektově orientovaný jazyk navržený v roce 1991 ve společnosti Sun Microsystems. Nyní je vyvíjen společností Oracle. Díky internetu se jazyk Java dostal na přední místo mezi programovacími jazyky. Kromě obecného zjednodušení webového programování jazyk Java také přinesl nové tzv. síťové programy nazývané aplety.

Aplet je speciální typ programu Java, který je určen pro přenos v internetu a který se automaticky spouští ve webovém prohlížeči s podporou technologie Java.

Nezávislost na operačním systému a hardwaru počítače je vyřešena způsobem kompilace programu. Java na výstupu neprodukuje spustitelný kód, ale bajtový kód. Bajtový kód se skládá z vysoce optimalizované sady instrukcí, které jsou určeny ke spuštění v systému runtime jazyka Java označovaném jako modul JVM (Java Virtual Machine).

3.2 Spring

Spring Framework [8] je nejpopulárnějším aplikačním vývojovým rámcem pro podnikovou platformu Java. Spring Framework je open source platforma Java. Základní verze Springu má kolem 2 MB. Základní funkce systému Spring Framework mohou být použity při vývoji libovolné aplikace Java, ale existují rozšíření pro vytváření webových aplikací na platformě Java EE. Cílem Spring frameworku je usnadnit vývoj J2EE a podporovat dobré programovací postupy tím, že umožňuje programovací model založený na POJO.

Inversion of Control (IOC) je obecný pojem a může být vyjádřen mnoha různými způsoby. Závislost injekce je jen jeden konkrétní příklad Inversion of Control.

Při psaní složité aplikace Java by třídy aplikací měly být co nejvíce nezávislé od jiných tříd jazyka Java, aby se zvýšila možnost opětovného použití těchto tříd a jejich testování nezávisle na jiných třídách při testování jednotky. Dependency Injection pomáhá při spojování těchto tříd dohromady a současně je udržuje nezávislé.

3.3 Javascript

Javascript [1] je interpretovaný, objektově orientovaný jazyk využívaný pro webové stránky. JavaScript běží na straně klienta, tedy v prohlížeči, až po stažení do počítače. JavaScript lze použít k návrhu či programování toho, jak se webové stránky chovají při výskytu události. JavaScript může fungovat jako procedurální i objektově orientovaný jazyk. Objekty jsou vytvářeny programově v JavaScriptu připojením metod a vlastností k jiným prázdným objektům za běhu, narozdíl od definic syntaktických tříd běžných v kompilovaných jazycích, jako jsou C++ a Java. Jakmile je objekt vybudován, může být použit jako návrh (nebo prototyp) pro vytváření podobných objektů.

3.4 PostgreSQL

PostgreSQL [2] je výkonný, otevřený objektově-relační databázový systém. Má více než 15 let aktivního vývoje a osvědčené architektury. Získal si silnou reputaci spolehlivosti, integrity dat a správnost. Spouští se na všech hlavních operačních systémech, včetně systémů Linux, UNIX (AIX, BSD, HP-UX, MacOS, Solaris) a Windows.

PostgreSQL umožňuje psát uložené procedury ve více než desítce programovacích jazyků, včetně Java, Perl, Pythonu, Ruby, C/C++ a vlastní PL/pgSQL, která je podobná PL/SQL systému Oracle. Součástí standardní knihovny funkcí jsou stovky vestavěných funkcí, které se pohybují od základních matematických a řetězových operací až po kryptografii a kompatibilitu Oracle. Triggery a uložené procedury mohou být zapsány v jazyku C a načteny do databáze jako knihovna, což umožňuje velkou flexibilitu při rozšíření jejich možností.

3.5 Hibernate

Hibernate [3] je open source projekt, jehož cílem je kompletní řešení problému správy persistence dat v jazyce Java. Persistence je jeden ze základních konceptů při vývoji aplikací. Pokud by program nezachovával data zadaná uživateli, při ukončení programu by data nebyla k dispozici. Kromě svého vlastního "nativního"api. Hibernate je také implementací specifikace Java persistence api (JPA). Jako takový jej lze snadno použít v jakémkoli prostředí podporujícím JPA včetně Java SE aplikací, Java EE aplikačních serverů, enterprise OSGi kontejnerů apod.

Hibernate umožňuje vytvářet persistentní třídy podle objektově orientovaných idiomů, včetně dědičnosti, polymorfismu, sdružování, kompozice a Java kolekcí. Hibernate nevyžaduje žádné rozhraní nebo základní třídy pro persistentní třídy a umožňuje persistenci jakékoli třídy nebo datové struktury.

3.6 HTML

HTML (Hypertextový značkovací jazyk) [5] html je základní jazyk používaný k vytváření dokumentů pro web a spolu s protokolem http (Hypertext Transfer Protocol) a URL (Uniform Resource Locator) je jedním ze tří hlavních protokolů na webu.

HTML se skládá ze sady předdefinovaných značek, které mohou návrháři webových stránek vložit do textu, aby zobrazili podrobnosti o tom, jak jsou webové stránky vykreslovány (tj. převedeny na konečný a snadno použitelný formulář) prostřednictvím webových prohlížečů. Mezi tyto podrobnosti patří odstavce, okraje, písma (včetně stylu a velikosti), sloupce, barvy (pozadí a text), odkazy, umístění obrázků, tok textu kolem obrázků, tabulky a prvky vstupních formulářů.

3.7 CSS

CSS [7] je jazyk popisující prezentaci webových stránek včetně barev, rozvržení a písem. Umožňuje přizpůsobení prezentace různým typům zařízení, jako jsou velké obrazovky, malé obrazovky

nebo tiskárny. CSS je nezávislá na HTML a může být použita s libovolným XML značkovacím jazykem. Oddělení HTML od CSS usnadňuje údržbu stránek, sdílení stylových listů mezi stránkami a přizpůsobení stránek různým prostředím. Toto je označováno jako oddělení struktury (nebo obsahu) od prezentace.

3.8 RabbitMQ

Rabbitmq [4] je open source message broker software (občas nazývaný message-oriented middleware), který původně implementoval pokročilý protokol pro fronty zpráv (amqp) a od té doby byl rozšířen o plug-in architekturu podporující Streaming Text Oriented Messaging Protocol (STOMP) mqtt a další protokoly. Zprávy umožňují propojení aplikací a mohou tak být součástí větší aplikace. Posílání zpráv je asynchronní, odděluje aplikace oddělením odesílání a přijímání dat.

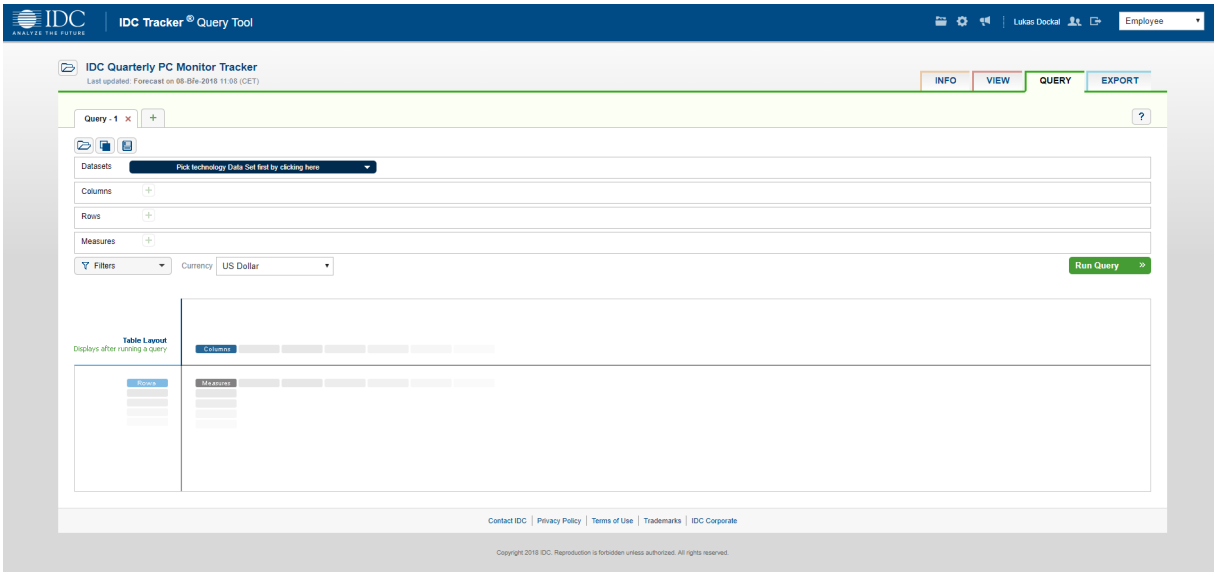
4 Práce na projektu Query Tool

4.1 Popis projektu

Query Tool je webová aplikace, která je schopna rychle reagovat na žádost o publikovaná data. Umožňuje provádět dotazy na jakýkoliv soubor dat, který si zákazníci zakoupili. Zákazníci mají také k dispozici základní panel se spoustou již předem vytvořených analýz a vizualizací dat za pomocí grafů. Dále Query Tool disponuje schopností exportovat požadovaná data do různých formátů jako jsou například csv, xlsx, ppt.

Aplikace je rozdělena na dvě části a to na administraci, kde mají přístup jen oprávnění uživatelé a na klientskou část, která je určená především zákazníkům, kteří mají možnost provádět dotazy na data, která mají zakoupena.

Query Tool je napsán v programovacím jazyku Java, kde hlavní komponentu celé aplikace tvoří Spring Framwork. Pro práci s velkými daty slouží technologie Apache Lucene. Pomocí technologie Apache Lucene se při publikaci data indexují přímo na disk a jsou nad nimi vytvořeny indexy, které pomáhají k rychlému přístupu k těmto datům. Klientská část vyžívá technologie JSP společně s javaScriptem, JQuery, HTML5 a CSS. Ukázka UI rozhraní apliakce Query Tool (Obrázek č.3).



Obrázek 3: Ukázka UI aplikace Query Tool

4.2 Blokování podezřelých uživatelů

Popis úkolu

Cílem tohoto úkolu bylo přidat do administrátorské části Query Tool aplikace možnost zablokování jednotlivých uživatelů aplikace na základě jejich emailové adresy.

Hlavní motivací pro tento úkol bylo poskytnout administrátorům Query Tool aplikace možnost zablokovat uživatele v případě zjištění podezřelé aktivity ze souborů aplikace typu log. Podezřelou aktivitou se rozumí například nadměrný export dat. Toto blokování může sloužit i jako prevence při zcizení uživatelského účtu.

Řešení úkolu

Tento úkol jsem rozdělil na dvě části. První část úkolu se týkala změn a úprav na straně serveru a druhá část je zaměřena na realizaci zobrazovací části, která bude sloužit k přidání uživatele na základě emailové adresy do seznamu blokovaných uživatelů.

V první části úkolu jsem musel vyřešit to, na základě jaké informace budeme uživatele blokovat. Jelikož údaje o uživatelích nejsou implicitně ukládány na straně Query Tool aplikace, je potřeba tyto informace nějakým způsobem ukládat. Vytvořil jsem tedy separátní tabulku v databázi, která obsahovala pole “email”, které slouží k evidenci zablokovaných uživatelů.

Následně bylo potřeba pro zablokovaný email odepřít přístup do apliakce Query Tool. Toto jsem realizoval vytvořením nové třídy s názvem UserKillSwitchFilter, která dědí z třídy Filter. V nově vytvořené třídě jsem přepsal metodu doFilter(), ve které si pomocí statické metody SecurityUtil.getCurrentUser() načtu aktuálního uživatele a ověřím, zda-li aktuální email uživatele není na seznamu blokovaných uživatelů. Pokud je email blokovaný, proběhne přesměrování na stránku s chybovou hláškou.

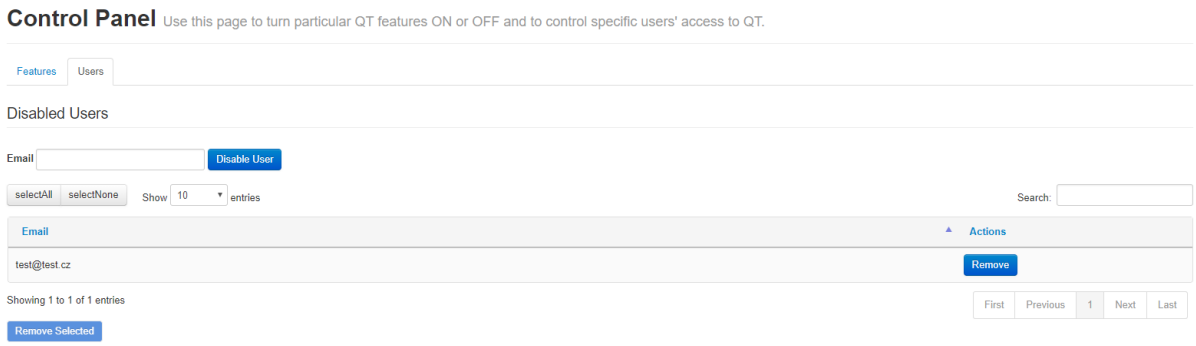
Dalším krokem bylo potřeba přidat do kontroleru metody, které budou volat na základě interakce administrátora s ui rozhraním. Ukázka metody kontroleru pro zablokování emailu (Ukázka kódu č.1).

Posledním krokem řešení tohoto úkolu bylo vytvoření jednoduchého uživatelského rozhraní, které se skládá ze dvou komponent. První komponentou je tabulka, která zobrazuje seznam blokovaných uživatelů společně s možností jednoduchého filtrovaní podle emailu. Další komponentou je formulář, který obsahuje vstupní pole pro blokovaný email a tlačítko sloužící k přidání uživatelského emailu mezi blokované. Ve formuláři jsem nastavil validaci emailu spolu s nutným potvrzením od uživatele s tím, že chce skutečně daný email zablokovat. Ukázka uživatelského rozhraní (Obrázek č.4).

Problémy při řešení úkolu

Problém, kterému jsem musel čelit je ten, že při jakémkoli požadavku na aplikaci dojde k dotazům na databázi, což není příliš efektivní. Tento problém jsem vyřešil tak, že jsem využil anotaci @Cacheable, která slouží k uložení výsledku vrácené metodou do paměti typu cache viz.

(Obrázek č.2). Tyto uložené výsledky je však důležité opět z cache vymazat, dojde-li například k odstranění blokovaného emailu, který je uložen v paměti cache. Toto je realizované pomocí anotace @QTCacheEvict např. na metodu removeDisabledUser(DisabledUser disabledUser) viz. (Ukázka kódu č.3), která se stará o odstranění blokovaného emailu a uložených výsledků z paměti.



Obrázek 4: Ukázka UI rozhraní pro blokování uživatelů

```
@RequestMapping(value = "disableUser", method = RequestMethod.POST)
@ResponseBody
public JSONResponse disableUser(@RequestParam(value = "email") String email) {

JSONResponse jsonResponse = new JSONResponse(true);
DisabledUser disabledUser;
if (StringUtils.isNotBlank(email)) {

disabledUser = settingsService.findDisabledUserByEmail(email);

if (disabledUser != null) {
handleError(jsonResponse, "User is already disabled");
} else {
disabledUser = new DisabledUser();
disabledUser.setEmail(email);
baseService.save(disabledUser);
}
} else {
handleError(jsonResponse, "Email is empty");
}
```

```
return jsonResponse;
}

private JSONResponse handleError(JSONResponse response, String message) {
response.setSuccess(false);
response.addErrorMessage(message);
return response;
}
```

Výpis 1: Metoda kontroleru pro zablokovaní eamilu

```
@Cacheable(value = CacheConstants.DISABLED_USERS, key = "'" + CacheConstants.
    DISABLED_USERS + "_'.concat(#email)", unless = "#result == null")
@Override
public boolean isUserDisabled(String email) {
   return settingsDAO.isUserDisabled(email);
}
```

Výpis 2: Ukládaní výsledku metody do cache

```
@QTCacheEvict(value = {CacheConstants.DISABLED_USERS}, key = "disabledUser.
    email")
@Override
public void removeDisabledUser(DisabledUser disabledUser) {
   settingsDAO.removeDisabledUser(disabledUser);
}

@QTCacheEvict(value = {CacheConstants.DISABLED_USERS}, key = "disabledUser.
    email")
@Override
public void disableUser(DisabledUser disabledUser) {
   settingsDAO.disableUser(disabledUser);
}
```

Výpis 3: Odstranění vysledku metody z cache

4.3 Omezení přístupu do apliakce Query Tool

Popis úkolu

V aplikaci Query Tool existuje funkcionalita, která uživatelům s patřičným oprávněním dovoluje přihlásit se za jiného uživatele bez nutnosti znalosti jeho hesla. Tato funkcionalita slouží

především k testování a simulaci podmínek, které měl daný uživatel v případě nějaké chyby. Cílem tohoto úkolu bylo zamezit přihlášení uživatelům, kteří jsou v databázi označeni jako neaktivní a také zamezit použití funkcionality přihlásit se za neaktivního uživatele.

Řešení úkolu

Nejdříve bylo nutné načíst údaje o uživateli pomocí REST api z IDC.com, protože údaje o uživatelích nejsou implicitně uloženy na straně aplikace Query Tool. Dále bylo potřeba zjistit na základě jakého atributu se dá zjistit, že je uživatel neaktivní. Jeden z atributů, který je součástí JSON objektu je atribut status. Atribut status je objekt, který obsahuje id stavu a jeho název. Tento atribut určuje, zda je uživatel aktivní či neaktivní. Atribut jsem přidal do třídy IdcUserDeatil. Následně už zbývalo upravit metodu loadUserByUsername(String username) viz. (ukázka kódu č.4) ve třídě AuthenticationUserDetailsServiceImpl o kontrolu zda status uživatele, který se snaží o přihlášení nebo někdo se za uživatele snaží přihlásit, není neaktivní. V případě, že uživatel je neaktivní, přihlášení se nezdaří a tento je informován o nezdařeném pokusu o přihlášení.

```
 public UserDetails loadUserByUsername(String username) throws
     UsernameNotFoundException {
User user = userSecurityService.getIdcUserDetails(username);

// If user is archived disable authentication
if (user.getStatus().getId() == 3) {
UserDetails userDetails;
String password = user.getPassword() == null ? "" : user.getPassword();
userDetails = new org.springframework.security.core.userdetails.User(user.
    getUsername(), password, false, true, true, true, user.getAuthorities());

return userDetails;
} else {
logCustomerProfiles(user);
return user;
}
}
```

Výpis 4: Metoda volaná při přihlašovaní uživatele

4.4 Zakázaní publikace pro dané prostředí

Popis úkolu

Nová data, která jsou vytvořena v aplikaci zvané Ferda a je potřeba dostat do aplikace Query Tool, kde jsou následně zpracovávána a poskytována zákazníkům. Celý tento proces se nazývá publikace. Využívá se RabbitMQ a skrze něj se pošle zpráva o nových datech. Na straně aplikace Query Tool je tato zpráva zachycena a poté dále zpracována. V hlavičce publikační zprávy je typ dat, formát a obsah, jehož struktura je popsaná ve formátu XML.

Cílem tohoto úkolu je zamezení publikace dat na konkrétní prostředí. Například není dobré, aby bylo možné publikovat zprávu z prostředí DEV na prostředí PROD.

Řešení úkolu

Nejprve bylo nutné zajistit, aby publikační zpráva obsahovala údaj o prostředí, z jakého byla odeslána. Toto bylo vyřešeno na straně aplikace Ferda, kterou má na starosti jiný tým. Na straně apliakce Query Tool bylo potřeba zabezpečit, aby nová hlavička nesoucí informaci o prostředí, byla ve zprávě zachována, protože rabbitmq ve výchozím nastavení odebírá definované hlavičky (tedy pokud není nakonfigurován, aby je ponechával).

V prvním kroku bylo nutné někde uchovávat informaci o povolené publikaci z prostředí. Zde jsem využil konfigurační soubor (Obrázek č.6), který obsahuje jednotlivá prostředí. Přidal jsem nový parametr, který obsahuje povolená prostředí oddělená čárkou v případě, že bude nutné povolit na konkrétním prostředí více příchozích publikací z různých prostředí.

Dále bylo zapotřebí upravit třídu BaseMessagingConsumer a metodu handleMessage (Ukázka kódu č.5), která se volá v okamžiku kdy je do fronty zaslána zpráva

```
 @Value("${qt.publish.allowed_environment}")
private List<String> allowedEnvironments;

@Override
public void handleMessage(Message message) {
try {
PublishMessageEnvironment publishMessageEnvironment = PublishMessageEnvironment
    .getPublishMessageEnvironment(String.valueOf(message.getHeaders().get(
    CommonConstants.ENVIRONMENT_HEADER_PROP)));

if (!allowedEnvironments.contains(publishMessageEnvironment.getName())) {
throw new MessageEnvironmentIsNotAllowed(publishMessageEnvironment);
}
```

```
logger.info("Message accepted by base messaging consumer: {}, publish message
    environment: {}", getParametrizedType(), publishMessageEnvironment.getName
    ());
acceptedMessage = message.getPayload().toString();
parametrizedType = getParametrizedType();
MT unmarshalledMessage = (MT) SerializationUtil.deserializeXMLMessage(
    acceptedMessage, parametrizedType);
processMessage(unmarshalledMessage, publishMessageEnvironment);

}
}
```

Výpis 5: Kontrola zda je prostředí povoleno

```
[DEV]
qt.publish.allowed_environment = QA,MIRROR,PROD
```

Výpis 6: Přidání nového parametru do konfiguračního souboru

4.5 Tvorba katalogu produktů

Popis úkolu

Tento úkol spočíval v přidání nové funkcionality do aplikace Query Tool, která bude mít za úkol poskytnou zákazníkům aplikace si zobrazit seznam všech produktů, které má zákazník koupené a ty, které si může koupit. Tato funkcionalita by se měla nacházet na úvodní stránce aplikace kam je uživatel přesměrován po přihlášení. Design nové funkcionality by měl být v podobném grafickém zpracování jako je již existující seznam zakoupených produktů na úvodní stránce aplikace.

Řešení úkolu

Úkol jsem řešil na dvě části. V první části úkolu bylo potřeba vhodně navrhnou design a umístění katalogu produktů na hlavní stránku. Při návrhu jsem vycházel z již existujícího designu zobrazování produktů na hlavní stránce. Vytvořil jsem tedy dvě záložky kde první z nich obsahovala seznam již koupených produktů a druhá obsahovala všechny produkty a po kliknutí na produkt se zobrazil seznam zakoupených i nezakoupených verzí barevně rozlišený. Při tvorbě UI jsem používal hlavně JSP k popisu struktury seznamu a CSS k aplikovaní vzhledu.

Druhá část úkolu spočívala v úpravách na straně serveru. Nejprve jsem vytvořil datovou strukturu pro ukládání produktů a jejich koupených či nekoupených verzí. Dále pak bylo zapotřebí vytvořit metodu viz. (Ukázka kódu č.7), která na základě parametru boughtDataSetsIds což je pole koupených verzí produktů, které má zákazník k dispozici. Na základě tohoto parametru metoda vytvoří daný katalog pro konkrétního uživatele, který bude zobrazen na hlavní stránce po přihlášení do aplikace.

```
private ProductCatalog createProductCatalogForTracker(Set<Long>
    boughtDataSetsIds) {
ProductCatalog productCatalog = new ProductCatalog();
List<DataSetDTO> dataSetDTOS = dataSetService.getAllDataSetDTOs();
for (DataSetDTO dataSetDTO : dataSetDTOS) {
boolean purchased = boughtDataSetsIds.contains(dataSetDTO.getDatasetId());
if (dataSetDTO.getDataSetType() != null &&StringUtils.isNotBlank(dataSetDTO.
    getLibraryName())&& (dataSetDTO.getDataSetType()==DataSetType.TRACKER ||
dataSetDTO.getDataSetType() == DataSetType.CHINA_TRACKER)) {
productCatalog.addProductCatalogItem(
dataSetDTO.getDataSetType().getName(),
dataSetDTO.getLibraryName(),
dataSetDTO.getName(),
purchased);
}
```

```
}
return productCatalog;
}
```

Výpis 7: Metoda vytvářející katalog produktů

4.6 Tvorba uživatelského rozhraní pro nastavení preferovaných měn

Popis úkolu

Cílem tohoto úkolu bylo navrhnout UI pro vybrání preferovaných měn. UI by mělo obsahovat list všech dostupných měn, které budou načítány z databáze. Dále si uživatel bude moci vybrat měny z listu všech měn, přičemž maximální počet měn, které si může vybrat, se bude načítat z databáze. UI by mělo být co nejvíce intuitivní, nemělo by tedy být složité na používání s ohledem na minimalizaci počtu kroků směřujících k vybrání měn a jejich uložení. Dalším důležitým faktorem je to, že u vybraných měn bude záležet na pořadí, v jakém budou zobrazeny.

Řešení úkolu

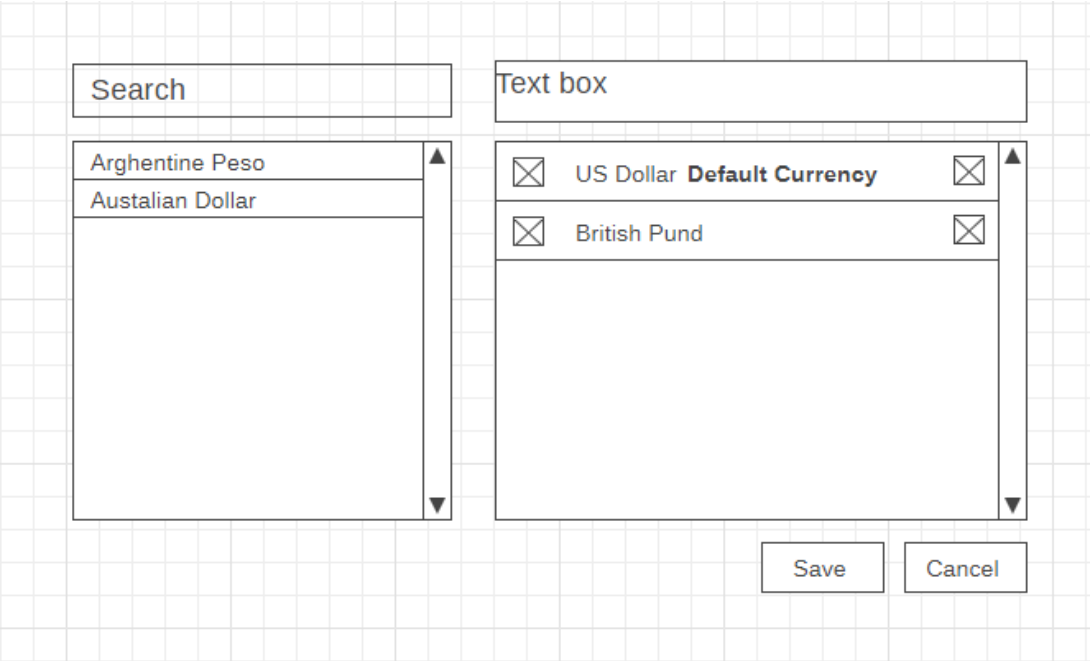
Při řešení úkolu jsem nejdříve začal s návrhem modelu (viz. Obrázek č.5). Model obsahuje dva listy. První list bude sloužit k vyobrazení všech měn, které budou načteny z databáze. Položky prvního listu budou na kliknutí přesunuty do druhé listu. Druhý list bude vyobrazovat vybrané měny, přičemž první měna v seznamu bude označena jako Default Currency. Z druhého listu bude také možné přidané měny odebírat pomocí tlačítka, které se bude nacházet na pravé straně názvu měny. Bude i možné měnit jejich pořadí pomocí uchopení ukazovacího zařízení (např. myši) a přesunutím na jiné místo v seznamu. Tato funkcionalita bude také minimalizovat počet akcí, které bude uživatel muset provádět za účelem změny pořadí vybraných měn.

Po vytvoření modelu, jak bude výsledné UI vypadat, jsem se pustil do hledání již existujícího řešení, které by splňovalo výše uvedené požadavky a bylo tak vhodným řešením tohoto úkolu. Bohužel, žádné z nalezených nesplňovalo podmínky zadání. Byl však nalezen JQuery plugin multiselect.js, který je pod licencí WTFPL, která umožňuje libovolnou modifikaci zdrojového kódu pro jakékoli použití. Vybraný plugin obsahoval ve své podstatě většinu hledané funkcionality až na pár věcí, které bylo potřeba implementovat.

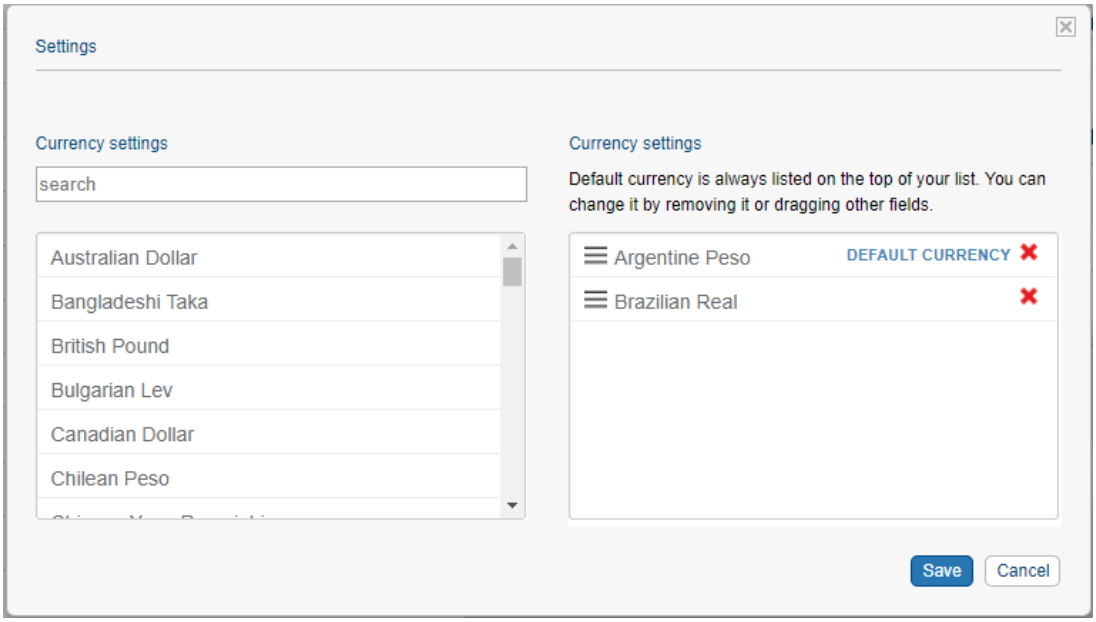
První z věcí, které bylo potřebné do pluginu přidat, byla možnost filtrovat položky v seznamu všech měn. To jsem řešil pomocí knihovny quicksearch.js.

Další funkcionalita, kterou bylo potřeba doplnit, bylo umožnění změny pořadí vybraných měn pomocí uchopení ukazovacího zařízení a přesunutím na jiné místo. Toto jsem realizoval pomocí knihovny jQuery UI, která je v aplikaci Query Tool běžně používána. Plugin obsahoval funkce typu callback. Pro vybrání a odebrání položky tedy bylo nutné přidat další, která bude reagovat na změnu pořadí. Rozhodl jsem se sjednotit tyto funkce do jedné, kterou jsem nazval afterUpdate. Tato metoda se bude volat při vybrání, odebrání, změně pořadí položky a bude vracet pole všech vybraných položek a událost, která tuto metodu spustila.

Další úpravy se týkaly převážně části HTML a CSS. Výsledná podoba komponenty je vidět na obrázku č.6.



Obrázek 5: Model uživatelského rozhraní pro nastavení preferovaných měn



Obrázek 6: Finální verze pro nastavení preferovaných měn

4.7 Zefektivnění transformací uživatelských dat

Popis úkolu

Během publikace dat v aplikaci zvané Ferda se mohou změnit id některých atributů v rámci data set verze, která jsou používána v uživatelských datech v aplikaci Query Tool. Transformace slouží k manuální úpravě neexistujících atributů v datech za nové. Doposud se transformace prováděla za pomoci vývojového týmu, protože bylo nutné do databáze přidat záznam o povodní hodnotě atributu a nové hodnotě ze stávající verze. Dále bylo nutné v uživatelském rozhraní pro transformaci dat zvolit příslušnou verzi dat, následně vybrat vložené deskriptory a provést transformaci s vybráním toho, nad kterými uživatelskými daty se má transformace provést.

Cílem tohoto úkolu bylo eliminovat manuální přidávání deskriptorů do databáze. Nové uživatelské rozhraní by mělo obsahovat formulář, který bude sloužit k přidávání nových deskriptorů a jejich odebírání u jednotlivých data set verzí. Dále bude formulář obsahovat základní validaci nově přidaných deskriptorů.

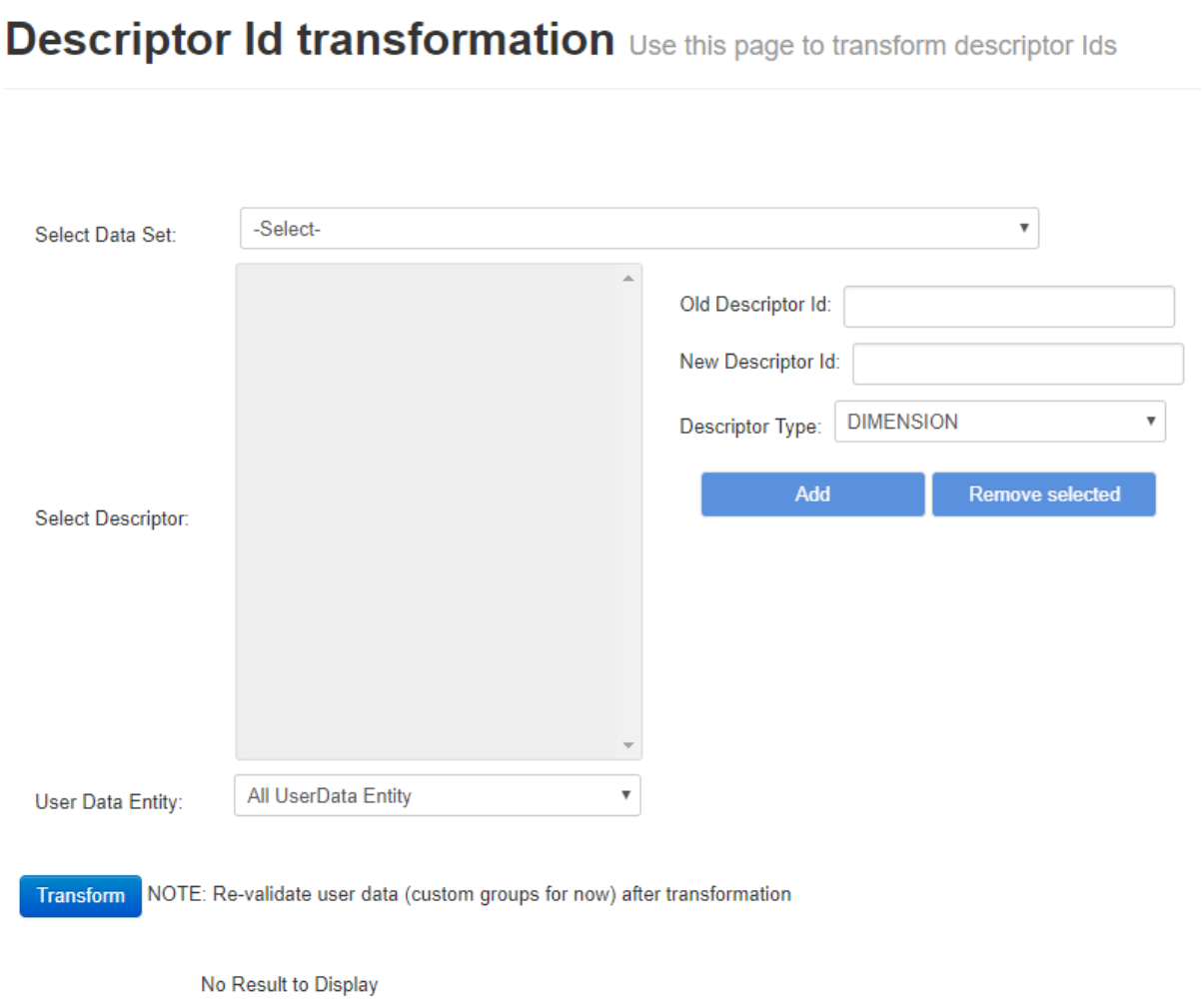
Řešení úkolu

V první fázi úkolu jsem vytvořil formulář (Obrázek č.7) se dvěma vstupními poli - jedno pro zadávání starého a nového názvu deskriptoru. Dalším prvek formuláře, který jsem přidal, bylo pole typu select box proto, aby bylo možné zvolit, o jaký typ deskriptoru se jedná, zda-li je nově přidávaný deskriptor dimenze či hodnota. Posledními a důležitými prvky formuláře jsou dvě tlačítka. Jedno slouží k přidání nového deskriptoru a další k odebírání jednoho nebo více deskriptorů. Formulář disponuje validací na straně klienta pomocí JavaScriptu, který po kliknutí na tlačítko “přidat” zkontroluje, zda-li název nového a starého deskriptoru je různý. V případě, že se jedná o stejné názvy hodnoty formuláře, tak se tyto nepošlou na sever a nedojde tak k uložení nového deskriptoru. V případě, že formulář není validní, uživatel je informován chybovou hláškou.

V druhé fázi řešení úkolu jsem přidal do servisní třídy metodu, která slouží k odebírání vybraných deskriptorů na základě jejich identifikátoru. Pro přidávání nového deskriptoru jsem využil již existující základní servisní třídu, která podporuje operace s objekty a jejich ukládání do databáze. Poté jsem vytvořil dva nové koncové body, které se provolávají po kliku na tlačítka “přidat” a “odebrat”. Koncový bod pro přidání deskriptoru obsahuje parametr descriptorJSON, který je typu řetězec a obsahuje reprezentaci objektu ve formátu JSON. DescriptorJSON obsahuje dataseId, dataSetId, oldDescriptorId, newDescriptorId, descriptorType. Tento řetězec je poté pomocí třídy ObjectMapper namapován na Java objekt DescriptorIdForm.

Poslední částí tohoto úkolu bylo vytvoření vlastního validátoru (Ukázka kódu č.9), který slouží k validaci objektu typu DescriptorIdForm. Validátor dědí ze třídy AbstractValidator a bylo nutné přepsat metodu validateInternal, která se volá metodou isValid a slouží k validování objektu.

Ukázka koncového bodu, který slouží k přidání nového deskriptoru (Ukázka kódu č.8).



Obrázek 7: Ukázka formuláře pro přidání dat

```
@ResponseBody
@RequestMapping(value = "descriptor/add", method = RequestMethod.POST)
public JSONResponse addDescriptor(@RequestParam("descriptorJSON") String
    descriptorJSON) {
   JSONResponse jsonResponse = new JSONResponse();
   jsonResponse.setSuccess(true);
   List<String> validationErrors = new ArrayList<>();
   DescriptorIdForm descriptorIdForm;

   try {
      ObjectMapper objectMapper = new ObjectMapper();
```

```
        descriptorIdForm = objectMapper.readValue(descriptorJSON,
            DescriptorIdForm.class);
    } catch (IOException e) {
        logger.error("Can not map json: {} to DescriptorIdForm object", e,
            descriptorJSON);
        jsonResponse.addErrorMessage("Descriptor can not be added");
        return jsonResponse;
    }

    if (!descriptorIdFormValidator.isValid(descriptorIdForm, validationErrors))
        {
        jsonResponse.setSuccess(false);
        jsonResponse.addAllErrorMessages(validationErrors);
        return jsonResponse;
    }

    DataSet<?> dataSet = baseService.getByID(DataSet.class, descriptorIdForm.
        getDataSetId());
    DataSetDescriptorType dataSetDescriptorType = DataSetDescriptorType.
        getDataSetDescriptorType(descriptorIdForm.getDescriptorType());
    DescriptorIdTransform descriptorIdTransform = new DescriptorIdTransform(
        dataSet, dataSetDescriptorType, descriptorIdForm.getOldDescriptorId(),
        descriptorIdForm.getNewDescriptorId());
    baseService.save(descriptorIdTransform);
    jsonResponse.setObject(descriptorIdTransform.getId());
    return jsonResponse;
}
```

Výpis 8: Metoda kontrolleru pro přidávání descriptorů

```
@Component
@Qualifier(value = "descriptorIdFormValidator")
public class DescriptorIdFormValidator extends AbstractValidator<
    DescriptorIdForm> {
    @Autowired
    private BaseService baseService;

    @Override
    protected boolean validateInternal(DescriptorIdForm objectToValidate) {
        return validateInternal(objectToValidate, new ArrayList<>());
```

```
    }

    @Override
    protected boolean validateInternal(DescriptorIdForm objectToValidate, List<
        String> validationMessages) {

    if (StringUtils.isBlank(objectToValidate.getNewDescriptorId())) {
       validationMessages.add("New Descriptor Id can not be empty");
    }

    if (StringUtils.isBlank(objectToValidate.getOldDescriptorId())) {
       validationMessages.add("Old Descriptor Id can not be empty");
    }

    if (DataSetDescriptorType.getDataSetDescriptorType(objectToValidate.
        getDescriptorType()) == null) {
       validationMessages.add("DataSet Descriptor Type is not exists");
    }

    if (baseService.getByID(DataSet.class, objectToValidate.getDataSetId()) ==
        null) {
       validationMessages.add("DataSet is not exists");
    }

    if(objectToValidate.getNewDescriptorId().equalsIgnoreCase(objectToValidate.
        getOldDescriptorId())) {
       validationMessages.add("New descriptor is same as old descriptor.");
    }

    return validationMessages.isEmpty();
    }
}
```

Výpis 9: Ukázka vlasního validátoru

5 Znalosti a dovednosti uplatněné v průběhu odborné praxe

Programovací jazyky II

- Tento kurz mi pomohl získat základní informace a znalosti v oblasti jazyku Java, které jsem uplatňoval po dobu celé praxe a hlouběji je rozvíjel.

Programování II

- Tento kurz mi poskytl znalosti týkající se objektově orientovaného programování, které jsem využíval v jazyce Java při řešení jednotlivých úkolů.

Úvod do databázových systémů

- Získané znalosti mi dopomohly zejména při řešení komplexních úkolů zaměřené na přepisování a optimalizaci SQL dotazů a návrhu databází.

Vývoj informačních systémů

- Znalosti získané po absolvování kurzu mi pomohly více se seznámit se základními návrhovými vzory a jejich aplikacemi. Zároveň jsem byl schopen rychleji pochopit architekturu aplikace Query Tool a také posoudit vhodnost použití návrhových vzorů při řešení úkolů.

6 Závěr

6.1 Scházející znalosti a dovednosti v průběhu odborné praxe

Scházející znalosti a dovednosti v průběhu odborné praxe Technologie, se kterou jsem přišel poprvé do kontaktu a rovněž jsem si potřeboval doplnit znalosti byla práce s RabbitMQ, kterou jsem využíval při řešení jednoho úkolu. Další technologie, se kterou jsem však pracoval denně a bylo tedy nutné prohloubit její znalost byla práce s frameworkem Spring a jeho součástí jako je Spring MVC.

Mezi dovednosti, které jsem potřeboval v neposlední řadě doplnit byly i znalosti anglického jazyka. Konkrétně mi scházela lepší dovednost konverzace, kterou jsem prakticky potřeboval každý den, protože veškerá komunikace, ať už týmová nebo v rámci jiných subjektů, probíhala v anglickém jazyce. Mezi další dovednost, která mi scházela, byla počáteční slabší orientace v projektu a architektuře, jelikož jsem do této doby měl pouze zkušenosti s menšími projekty a to především v rámci školy.

6.2 Celkové zhodnocení odborné praxe a dosažených výsledků

V prvních týdnech jsem byl seznámen s architekturou aplikace Query Tool a s používanými technologiemi. Mezi ně patří například AWS, systém pro správu sestavování aplikací Apache maven, představení technologie Apache Lucene, frameworok pro vývoj JavaEE aplikací Spring. Po úvodním seznámení s technologiemi a aplikací Query Tool jsem pracoval na menších úkolech, které sloužily k postupnému zapojování se do týmu a k přípravě práce na složitějších zadáních.

Celkově odbornou praxi hodnotím velmi kladně. Během dvou semestrů, které jsem strávil v IDC, jsem si mohl rozšířit znalosti a to jak z oblasti vývoje webových aplikací a databází, tak i v komunikaci v anglickém jazyce. Během celé praxe jsem si vyzkoušel práci v týmu a měl jsem tak příležitost zkombinovat znalosti a dovednosti nabyté za doby studia a využít je v praxi.

Během absolvování odborné praxe jsem si musel doplnit chybějící znalosti, ale díky velké podpoře ze strany kolegů byla tato praxe velice dobrou první pracovní zkušeností.

Literatura

[1] About JavaScript - JavaScript MDN. About JavaScript [online]. [cit. 2018-02-17]. Dostupné z: http://www-cs-faculty.stanford.edu/~{}uno/abcde.html

[2] PostgreSQL: About. About [online]. [cit. 2018-02-17]. Dostupné z: https://www.postgresql.org/about/

[3] BAUER, CHRISTIAN a GAVIN KING. Hibernate in Action. Manning Publications Co, 2005. ISBN 1932394-15-X.

[4] RabbitMQ - Messaging that just works [online]. [cit. 2018-02-17]. Dostupné z: https://www.rabbitmq.com

[5] Introduction to HTML - Learn web development. Introduction to HTML [online]. [cit. 2018-02-17]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML

[6] SCHILDT, Herbert a Jakub GONER. Java 8 Výukový kurz. Brno: Computer press, 2016. ISBN ISBN 978-80-251-4665-1.

[7] HTML & CSS - W3C: What is CSS? [online]. [cit. 2018-03-24]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss

[8] Spring Framework Overview [online]. [cit. 2018-04-13]. Dostupné z: https://www.tutorialspoint.com/spring/spring_overview.htm

[9] About IDC: International Data Corporation [online]. [cit. 2018-03-25]. [cit. 2018-02-25]. Dostupné z: https://www.idc.com/about